

**MACHINE LEARNING**

**LAB MANUAL**

**5th SEMESTER, COMMON FOR ALL PROGRAMS**

**OF**

**SCHOOL OF ENGINEERING & TECHNOLOGY**

1. **Decision Tree Classifier**

**Implement and demonstrate a Decision Tree Classifier to classify the instances of dataset. Display the classification results. Also, try the same algorithm to classify the instances for any given medical diagnosis dataset.**

1. Data Pre-Processing Step:

# importing libraries

import numpy as nm

import matplotlib.pyplot as mtp

import pandas as pd

#importing datasets

data\_set= pd.read\_csv('user\_data.csv')

#Extracting Independent and dependent Variable

x= data\_set.iloc[:, [2,3]].values

y= data\_set.iloc[:, 4].values

# Splitting the dataset into training and test set.

from sklearn.model\_selection import train\_test\_split

x\_train, x\_test, y\_train, y\_test= train\_test\_split(x, y, test\_size= 0.25, random\_state=0)

#feature Scaling

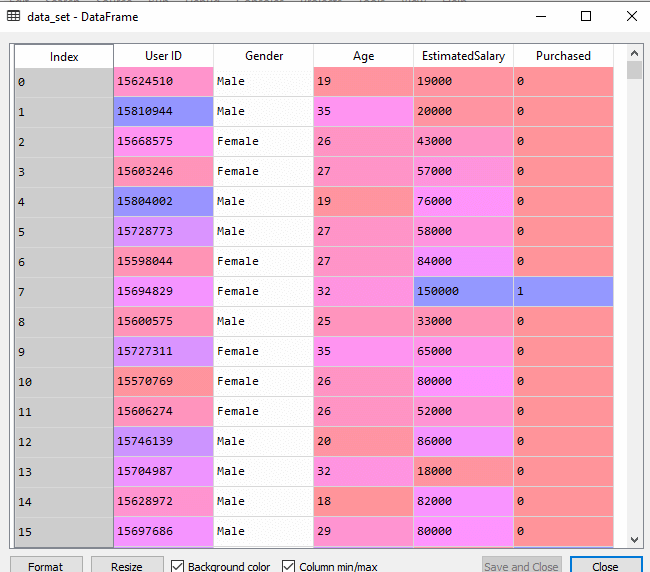
from sklearn.preprocessing import StandardScaler

st\_x= StandardScaler()

x\_train= st\_x.fit\_transform(x\_train)

x\_test= st\_x.transform(x\_test)

output:



### 2. Fitting a Decision-Tree algorithm to the Training set

#Fitting Decision Tree classifier to the training set

From sklearn.tree import DecisionTreeClassifier

classifier= DecisionTreeClassifier(criterion='entropy', random\_state=0)

classifier.fit(x\_train, y\_train)

Output:

Out[8]:

DecisionTreeClassifier(class\_weight=None, criterion='entropy', max\_depth=None,

max\_features=None, max\_leaf\_nodes=None,

min\_impurity\_decrease=0.0, min\_impurity\_split=None,

min\_samples\_leaf=1, min\_samples\_split=2,

min\_weight\_fraction\_leaf=0.0, presort=False,

random\_state=0, splitter='best')

### 3. Predicting the test result

### #Predicting the test set result

### y\_pred= classifier.predict(x\_test)

### **Output:**

### Decision Tree Classification Algorithm

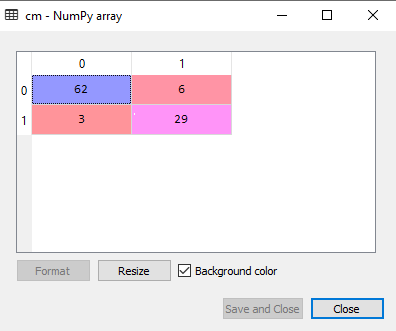
### 4. Test accuracy of the result (Creation of Confusion matrix)

#Creating the Confusion matrix

from sklearn.metrics import confusion\_matrix

cm= confusion\_matrix(y\_test, y\_pred)

Output:



### 5. Visualizing the training set result:

#Visulaizing the trianing set result

from matplotlib.colors import ListedColormap

x\_set, y\_set = x\_train, y\_train

x1, x2 = nm.meshgrid(nm.arange(start = x\_set[:, 0].min() - 1, stop = x\_set[:, 0].max() + 1, step =0.01),

nm.arange(start = x\_set[:, 1].min() - 1, stop = x\_set[:, 1].max() + 1, step = 0.01))

mtp.contourf(x1, x2, classifier.predict(nm.array([x1.ravel(), x2.ravel()]).T).reshape(x1.shape),

alpha = 0.75, cmap = ListedColormap(('purple','green' )))

mtp.xlim(x1.min(), x1.max())

mtp.ylim(x2.min(), x2.max())

fori, j in enumerate(nm.unique(y\_set)):

mtp.scatter(x\_set[y\_set == j, 0], x\_set[y\_set == j, 1],

c = ListedColormap(('purple', 'green'))(i), label = j)

mtp.title('Decision Tree Algorithm (Training set)')

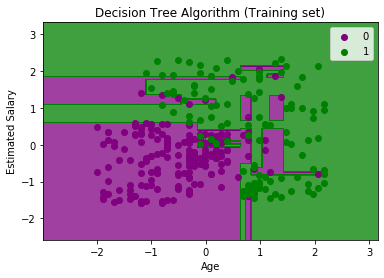
mtp.xlabel('Age')

mtp.ylabel('Estimated Salary')

mtp.legend()

mtp.show()

**Output:**



### 6. Visualizing the test set result

#Visulaizing the test set result

from matplotlib.colors import ListedColormap

x\_set, y\_set = x\_test, y\_test

x1, x2 = nm.meshgrid(nm.arange(start = x\_set[:, 0].min() - 1, stop = x\_set[:, 0].max() + 1, step =0.01),

nm.arange(start = x\_set[:, 1].min() - 1, stop = x\_set[:, 1].max() + 1, step = 0.01))

mtp.contourf(x1, x2, classifier.predict(nm.array([x1.ravel(), x2.ravel()]).T).reshape(x1.shape),

alpha = 0.75, cmap = ListedColormap(('purple','green' )))

mtp.xlim(x1.min(), x1.max())

mtp.ylim(x2.min(), x2.max())

fori, j in enumerate(nm.unique(y\_set)):

mtp.scatter(x\_set[y\_set == j, 0], x\_set[y\_set == j, 1],

c = ListedColormap(('purple', 'green'))(i), label = j)

mtp.title('Decision Tree Algorithm(Test set)')

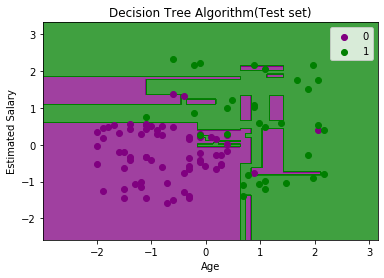
mtp.xlabel('Age')

mtp.ylabel('Estimated Salary')

mtp.legend()

mtp.show()

Output:



1. **Feature extraction using Principal Component Analysis (PCA)**

**Implement and demonstrate the Principal Component Analysis algorithm for dimensionality reduction for any dataset**

**Step 1:** We will import the libraries.

import numpy as nmp

import matplotlib.pyplot as mpltl

import pandas as pnd

**Step 2:** We will import the dataset (wine.csv)

DS = pnd.read\_csv('Wine.csv')

# Now, we will distribute the dataset into two components "X" and "Y"

X = DS.iloc[: , 0:13].values

Y = DS.iloc[: , 13].values

**Step 3:** In this step, we will split the dataset into the training set and testing set.

from sklearn.model\_selection import train\_test\_split as tts

X\_train, X\_test, Y\_train, Y\_test = tts(X, Y, test\_size = 0.2, random\_state = 0)

**Step 4:** Now, we will Feature Scaling.

from sklearn.preprocessing import StandardScaler as SS

SC = SS()

X\_train = SC.fit\_transform(X\_train)

X\_test = SC.transform(X\_test)

**Step 5:** Then, Apply the PCA function

from sklearn.decomposition import PCA

PCa = PCA (n\_components = 1)

X\_train = PCa.fit\_transform(X\_train)

X\_test = PCa.transform(X\_test)

explained\_variance = PCa.explained\_variance\_ratio\_

**Step 6:** Now, we will fit Logistic Regression for the training set

from sklearn.linear\_model import LogisticRegression as LR

classifier\_1 = LR (random\_state = 0)

classifier\_1.fit(X\_train, Y\_train)

**Output:**

LogisticRegression(random\_state=0

**Step 7:** Here, we will predict the testing set result:  
Y\_pred = classifier\_1.predict(X\_test))

**Step 8:** We will create the confusion matrix

from sklearn.metrics import confusion\_matrix as CM

cm = CM (Y\_test, Y\_pred)

**Step 9:** Then, predict the result of the training set:

from matplotlib.colors import ListedColormap as LCM

X\_set, Y\_set = X\_train, Y\_train

X\_1, X\_2 = nmp.meshgrid(nmp.arange(start = X\_set[:, 0].min() - 1,

stop = X\_set[: , 0].max() + 1, step = 0.01),

nmp.arange(start = X\_set[: , 1].min() - 1,

stop = X\_set[: , 1].max() + 1, step = 0.01))

mpltl.contourf(X\_1, X\_2, classifier\_1.predict(nmp.array([X\_1.ravel(),

X\_2.ravel()]).T).reshape(X\_1.shape), alpha = 0.75,

cmap = LCM (('yellow', 'grey', 'green')))

mpltl.xlim (X\_1.min(), X\_1.max())

mpltl.ylim (X\_2.min(), X\_2.max())

for s, t in enumerate(nmp.unique(Y\_set)):

mpltl.scatter(X\_set[Y\_set == t, 0], X\_set[Y\_set == t, 1],

c = LCM (('red', 'green', 'blue'))(s), label = t)

mpltl.title('Logistic Regression for Training set: ')

mpltl.xlabel ('PC\_1') # for X\_label

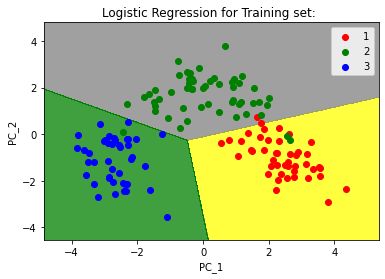
mpltl.ylabel ('PC\_2') # for Y\_label

mpltl.legend() # for showing legend

# show scatter plot

mpltl.show()

Output:



**Step 10:** At last, we will visualize the result of the testing set

from matplotlib.colors import ListedColormap as LCM

X\_set, Y\_set = X\_test, Y\_test

X\_1, X\_2 = nmp.meshgrid(nmp.arange(start = X\_set[: , 0].min() - 1,

stop = X\_set[: , 0].max() + 1, step = 0.01),

nmp.arange(start = X\_set[: , 1].min() - 1,

stop = X\_set[: , 1].max() + 1, step = 0.01))

mpltl.contourf(X\_1, X\_2, classifier\_1.predict(nmp.array([X\_1.ravel(),

X\_2.ravel()]).T).reshape(X\_1.shape), alpha = 0.75,

cmap = LCM(('pink', 'grey', 'aquamarine')))

mpltl.xlim(X\_1.min(), X\_1.max())

mpltl.ylim(X\_2.min(), X\_2.max())

for s, t in enumerate(nmp.unique(Y\_set)):

mpltl.scatter(X\_set[Y\_set == t, 0], X\_set[Y\_set == t, 1],

c = LCM(('red', 'green', 'blue'))(s), label = t)

# title for scatter plot

mpltl.title('Logistic Regression for Testing set')

mpltl.xlabel ('PC\_1') # for X\_label

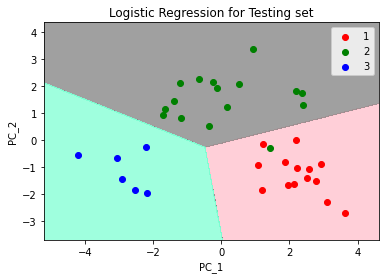
mpltl.ylabel ('PC\_2') # for Y\_label

mpltl.legend()

# show scatter plot

mpltl.show()

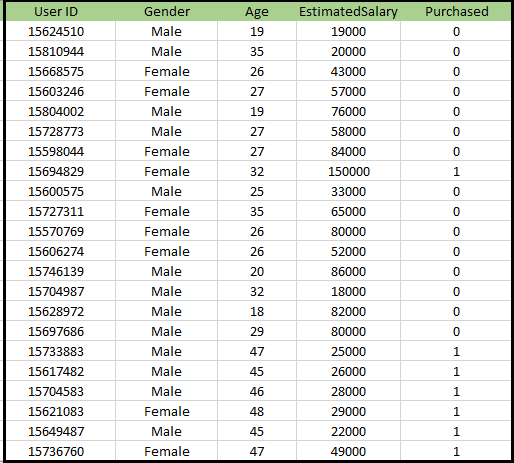
output:



**3. K nearest neighbour (KNN)**

**Implement and demonstrate the k-Nearest Neighbour algorithm (k-NN) to classify the iris data set. Display the Confusion matrix and classification report. Also, try the same algorithm of the social networks dataset to predict a customer can purchase an item or not.**

**Problem for K-NN Algorithm:** The dataset contains lots of information but the **Estimated Salary** and **Age** we will consider for the independent variable and the **Purchased variable** is for the dependent variable. Below is the dataset:



**Steps to implement the K-NN algorithm:**

* Data Pre-processing step
* Fitting the K-NN algorithm to the Training set
* Predicting the test result
* Test accuracy of the result(Creation of Confusion matrix)
* Visualizing the test set result.

Step 1: Data Pre-Processing Step:

The Data Pre-processing step will remain exactly the same as Logistic Regression. Below is the code for it:

# importing libraries

import numpy as nm

import matplotlib.pyplot as mtp

import pandas as pd

#importing datasets

data\_set= pd.read\_csv('user\_data.csv')

#Extracting Independent and dependent Variable

x= data\_set.iloc[:, [2,3]].values

y= data\_set.iloc[:, 4].values

# Splitting the dataset into training and test set.

from sklearn.model\_selection import train\_test\_split

x\_train, x\_test, y\_train, y\_test= train\_test\_split(x, y, test\_size= 0.25, random\_state=0)

#feature Scaling

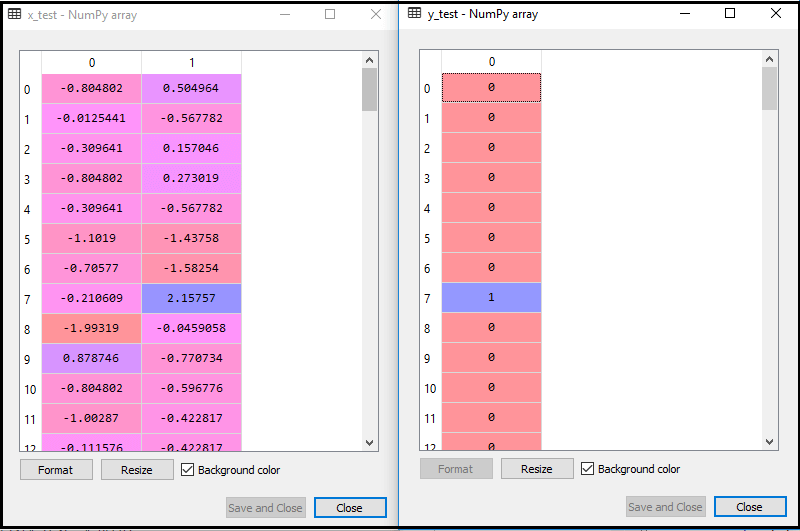
from sklearn.preprocessing import StandardScaler

st\_x= StandardScaler()

x\_train= st\_x.fit\_transform(x\_train)

x\_test= st\_x.transform(x\_test)

Ouput:



Step 2: Fitting the K-NN algorithm to the Training set

#Fitting K-NN classifier to the training set

from sklearn.neighbors import KNeighborsClassifier

classifier= KNeighborsClassifier(n\_neighbors=5, metric='minkowski', p=2 )

classifier.fit(x\_train, y\_train

Output:

Out[10]:

KNeighborsClassifier(algorithm='auto', leaf\_size=30, metric='minkowski',

metric\_params=None, n\_jobs=None, n\_neighbors=5, p=2,

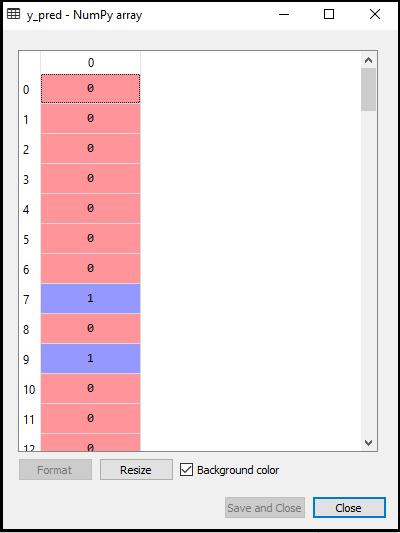
weights='uniform')

Step 3: Predicting the test result

#Predicting the test set result

y\_pred= classifier.predict(x\_test)

output:



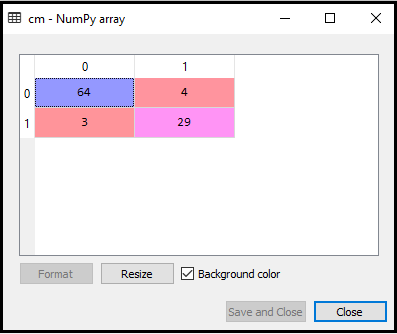
Step 4 : **Creating the Confusion Matrix:**

#Creating the Confusion matrix

from sklearn.metrics import confusion\_matrix

cm= confusion\_matrix(y\_test, y\_pred)

**Output:**



Step 5: **Visualizing the Training set result**

#Visulaizing the trianing set result

from matplotlib.colors import ListedColormap

x\_set, y\_set = x\_train, y\_train

x1, x2 = nm.meshgrid(nm.arange(start = x\_set[:, 0].min() - 1, stop = x\_set[:, 0].max() + 1, step =0.01),

nm.arange(start = x\_set[:, 1].min() - 1, stop = x\_set[:, 1].max() + 1, step = 0.01))

mtp.contourf(x1, x2, classifier.predict(nm.array([x1.ravel(), x2.ravel()]).T).reshape(x1.shape),

alpha = 0.75, cmap = ListedColormap(('red','green' )))

mtp.xlim(x1.min(), x1.max())

mtp.ylim(x2.min(), x2.max())

for i, j in enumerate(nm.unique(y\_set)):

mtp.scatter(x\_set[y\_set == j, 0], x\_set[y\_set == j, 1],

c = ListedColormap(('red', 'green'))(i), label = j)

mtp.title('K-NN Algorithm (Training set)')

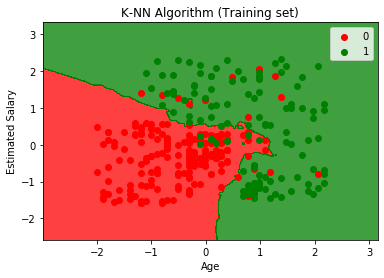
mtp.xlabel('Age')

mtp.ylabel('Estimated Salary')

mtp.legend()

mtp.show()

**Output:**



Step 6: **Visualizing the Test set result:**

#Visualizing the test set result

from matplotlib.colors import ListedColormap

x\_set, y\_set = x\_test, y\_test

x1, x2 = nm.meshgrid(nm.arange(start = x\_set[:, 0].min() - 1, stop = x\_set[:, 0].max() + 1, step =0.01),

nm.arange(start = x\_set[:, 1].min() - 1, stop = x\_set[:, 1].max() + 1, step = 0.01))

mtp.contourf(x1, x2, classifier.predict(nm.array([x1.ravel(), x2.ravel()]).T).reshape(x1.shape),

alpha = 0.75, cmap = ListedColormap(('red','green' )))

mtp.xlim(x1.min(), x1.max())

mtp.ylim(x2.min(), x2.max())

for i, j in enumerate(nm.unique(y\_set)):

mtp.scatter(x\_set[y\_set == j, 0], x\_set[y\_set == j, 1],

c = ListedColormap(('red', 'green'))(i), label = j)

mtp.title('K-NN algorithm(Test set)')

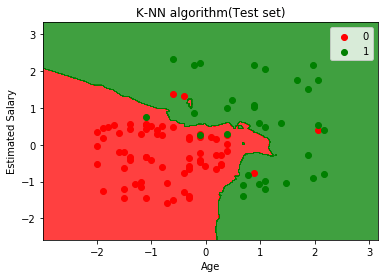
mtp.xlabel('Age')

mtp.ylabel('Estimated Salary')

mtp.legend()

mtp.show()

output:



**4. Support Vector Machine (SVM)**

**Implement and demonstrate a Support vector machine classifier to classify the instances of any dataset. Display the classification results. Also, try the same algorithm to classify the instances for any given dataset.**

**Python Implementation of Support Vector Machine**

Now we will implement the SVM algorithm using Python. Here we will use the same dataset **user\_data**, which we have used in Logistic regression and KNN classification.

Step 1: Data Pre-processing step

Till the Data pre-processing step, the code will remain the same. Below is the code:

#Data Pre-processing Step

# importing libraries

import numpy as nm

import matplotlib.pyplot as mtp

import pandas as pd

#importing datasets

data\_set= pd.read\_csv('user\_data.csv')

#Extracting Independent and dependent Variable

x= data\_set.iloc[:, [2,3]].values

y= data\_set.iloc[:, 4].values

# Splitting the dataset into training and test set.

from sklearn.model\_selection import train\_test\_split

x\_train, x\_test, y\_train, y\_test= train\_test\_split(x, y, test\_size= 0.25, random\_state=0)

#feature Scaling

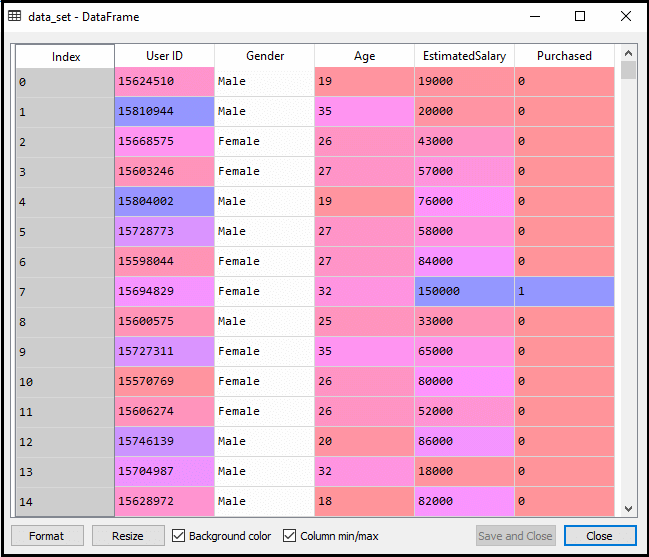
from sklearn.preprocessing import StandardScaler

st\_x= StandardScaler()

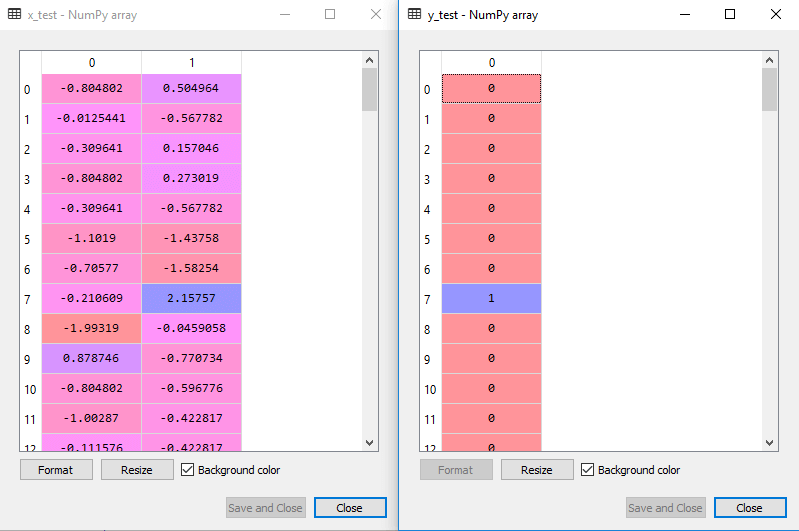
x\_train= st\_x.fit\_transform(x\_train)

x\_test= st\_x.transform(x\_test)

**output:**

****

The scaled output for the test set will be

****

Step 2: **Fitting the SVM classifier to the training set:**

from sklearn.svm import SVC # "Support vector classifier"

classifier = SVC(kernel='linear', random\_state=0)

classifier.fit(x\_train, y\_train)

**Output:**

Out[8]:

SVC(C=1.0, cache\_size=200, class\_weight=None, coef0=0.0,

decision\_function\_shape='ovr', degree=3, gamma='auto\_deprecated',

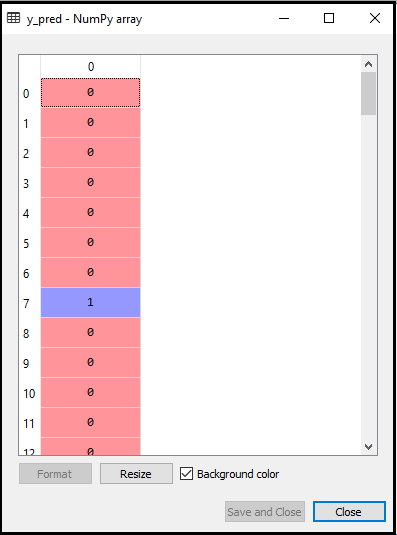
kernel='linear', max\_iter=-1, probability=False, random\_state=0,

shrinking=True, tol=0.001, verbose=False)

Step 3: **Predicting the test set result**

#Predicting the test set result

y\_pred= classifier.predict(x\_test)



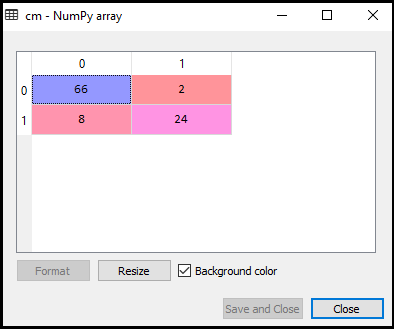
Step 4 : **Creating the confusion matrix**

#Creating the Confusion matrix

from sklearn.metrics import confusion\_matrix

cm= confusion\_matrix(y\_test, y\_pred)

Output:



Step 5: **Visualizing the training set result**

from matplotlib.colors import ListedColormap

x\_set, y\_set = x\_train, y\_train

x1, x2 = nm.meshgrid(nm.arange(start = x\_set[:, 0].min() - 1, stop = x\_set[:, 0].max() + 1, step =0.01),

nm.arange(start = x\_set[:, 1].min() - 1, stop = x\_set[:, 1].max() + 1, step = 0.01))

mtp.contourf(x1, x2, classifier.predict(nm.array([x1.ravel(), x2.ravel()]).T).reshape(x1.shape),

alpha = 0.75, cmap = ListedColormap(('red', 'green')))

mtp.xlim(x1.min(), x1.max())

mtp.ylim(x2.min(), x2.max())

for i, j in enumerate(nm.unique(y\_set)):

mtp.scatter(x\_set[y\_set == j, 0], x\_set[y\_set == j, 1],

c = ListedColormap(('red', 'green'))(i), label = j)

mtp.title('SVM classifier (Training set)')

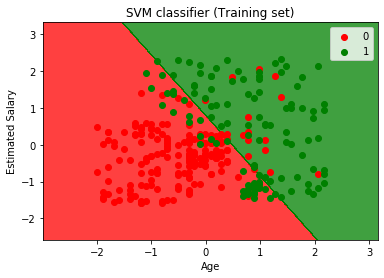
mtp.xlabel('Age')

mtp.ylabel('Estimated Salary')

mtp.legend()

mtp.show()

output:



Step 5: **Visualizing the test set result**

#Visulaizing the test set result

from matplotlib.colors import ListedColormap

x\_set, y\_set = x\_test, y\_test

x1, x2 = nm.meshgrid(nm.arange(start = x\_set[:, 0].min() - 1, stop = x\_set[:, 0].max() + 1, step =0.01),

nm.arange(start = x\_set[:, 1].min() - 1, stop = x\_set[:, 1].max() + 1, step = 0.01))

mtp.contourf(x1, x2, classifier.predict(nm.array([x1.ravel(), x2.ravel()]).T).reshape(x1.shape),

alpha = 0.75, cmap = ListedColormap(('red','green' )))

mtp.xlim(x1.min(), x1.max())

mtp.ylim(x2.min(), x2.max())

for i, j in enumerate(nm.unique(y\_set)):

mtp.scatter(x\_set[y\_set == j, 0], x\_set[y\_set == j, 1],

c = ListedColormap(('red', 'green'))(i), label = j)

mtp.title('SVM classifier (Test set)')

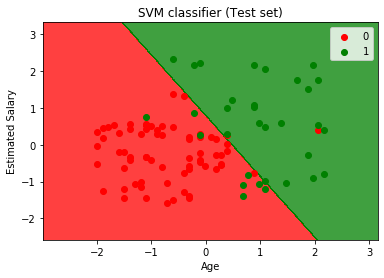
mtp.xlabel('Age')

mtp.ylabel('Estimated Salary')

mtp.legend()

mtp.show()

**Output:**



**5. Short Title: Regression**

**Implement and demonstrate linear regression and logistic regression algorithms for any given dataset(s). Visualize the results using graphs.**

**(Salary prediction, Price Prediction)**

**Problem Statement example for Simple Linear Regression:**

Here we are taking a dataset that has two variables: salary (dependent variable) and experience (Independent variable). The goals of this problem is:

* **We want to find out if there is any correlation between these two variables**
* **We will find the best fit line for the dataset.**
* **How the dependent variable is changing by changing the independent variable.**

**Step-1: Data Pre-processing**

First, we will import the three important libraries, which will help us for loading the dataset, plotting the graphs, and creating the Simple Linear Regression model.

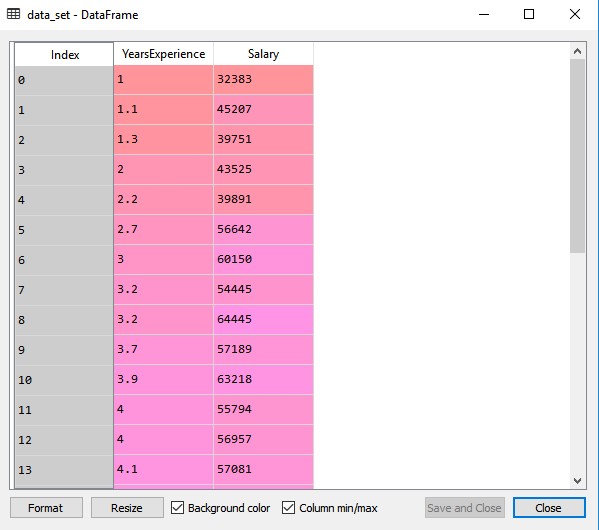
import numpy as nm

import matplotlib.pyplot as mtp

import pandas as pd

Next, we will load the dataset into our code:

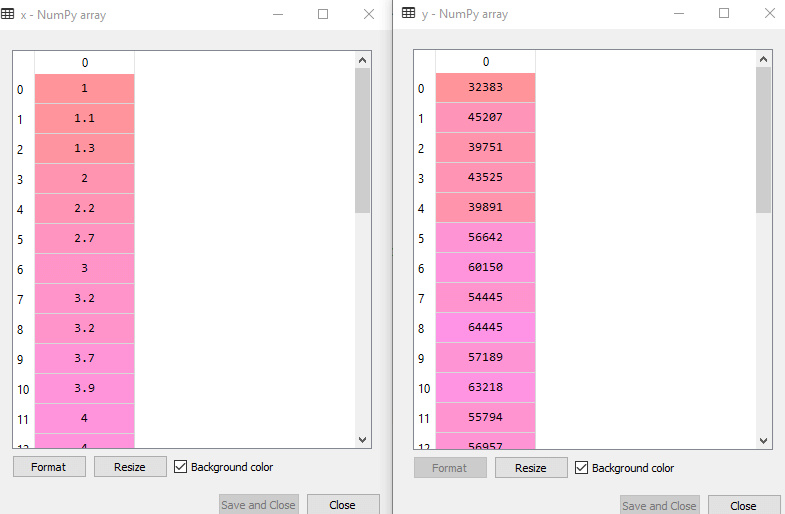
data\_set= pd.read\_csv('Salary\_Data.csv')



x= data\_set.iloc[:, :-1].values

y= data\_set.iloc[:, 1].values

output:

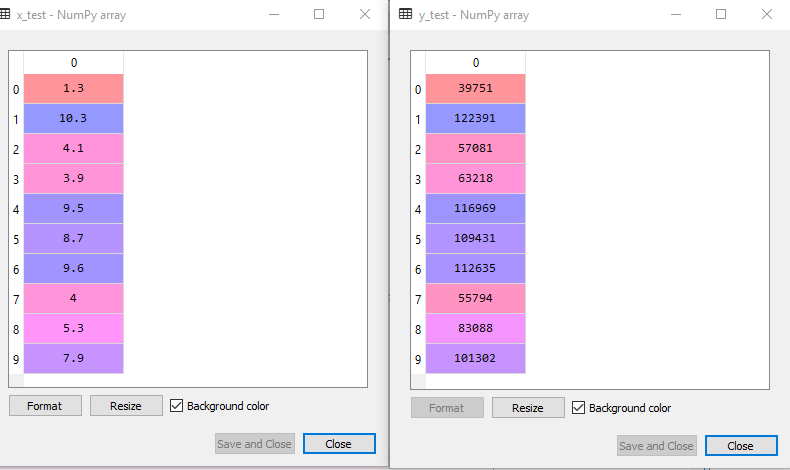


# Splitting the dataset into training and test set.

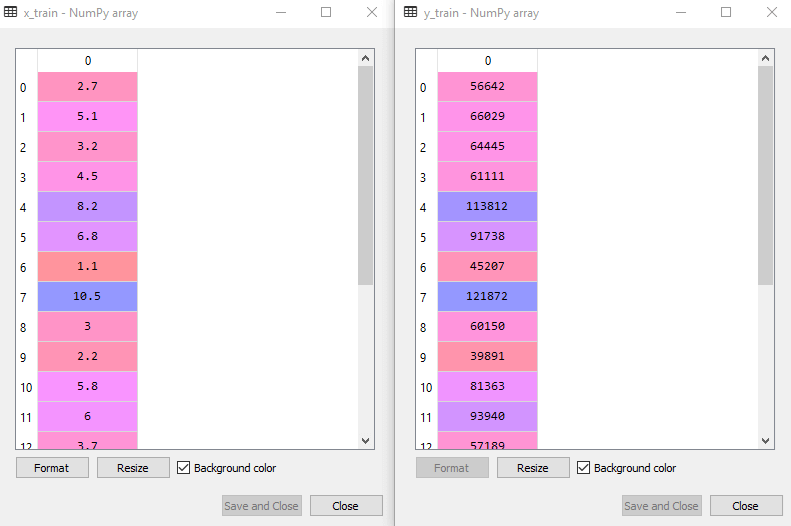
from sklearn.model\_selection import train\_test\_split

x\_train, x\_test, y\_train, y\_test= train\_test\_split(x, y, test\_size= 1/3, random\_state=0)

**Test-dataset:**



**Training Dataset:**



**Step-2: Fitting the Simple Linear Regression to the Training Set**

#Fitting the Simple Linear Regression model to the training dataset

from sklearn.linear\_model import LinearRegression

regressor= LinearRegression()

regressor.fit(x\_train, y\_train)

Output:

Out[7]: LinearRegression(copy\_X=True, fit\_intercept=True, n\_jobs=None, normalize=False)

**Step: 3. Prediction of test set result:**

#Prediction of Test and Training set result

y\_pred= regressor.predict(x\_test)

x\_pred= regressor.predict(x\_train)

**Step: 4. visualizing the Training set results:**

mtp.scatter(x\_train, y\_train, color="green")

mtp.plot(x\_train, x\_pred, color="red")

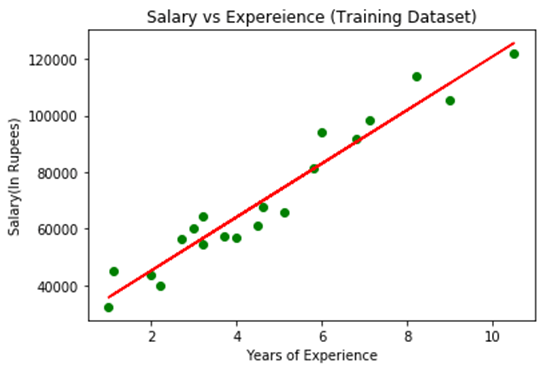
mtp.title("Salary vs Experience (Training Dataset)")

mtp.xlabel("Years of Experience")

mtp.ylabel("Salary(In Rupees)")

mtp.show()

**Output:**



**Step: 5. visualizing the Test set results**

#visualizing the Test set results

mtp.scatter(x\_test, y\_test, color="blue")

mtp.plot(x\_train, x\_pred, color="red")

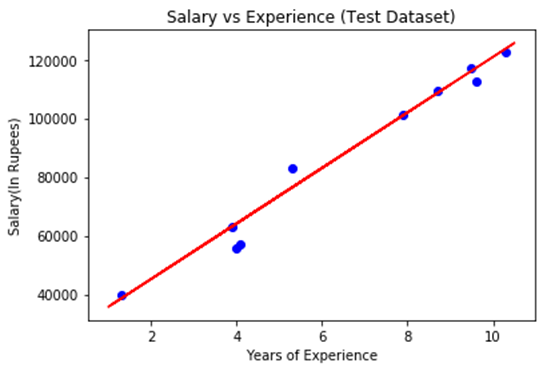
mtp.title("Salary vs Experience (Test Dataset)")

mtp.xlabel("Years of Experience")

mtp.ylabel("Salary(In Rupees)")

mtp.show()

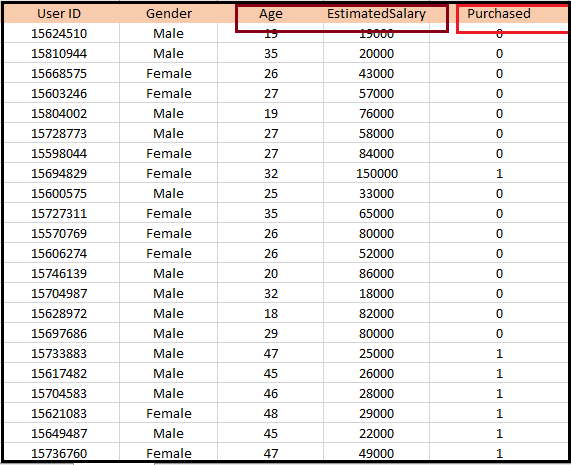
**Output:**



**Problem Statement example for Simple logistic Regression:**

There is a dataset given which contains the information of various users obtained from the social networking sites. There is a car making company that has recently launched a new SUV car. So the company wanted to check how many users from the dataset, wants to purchase the car.

For this problem, we will build a Machine Learning model using the Logistic regression algorithm. The dataset is shown in the below image. In this problem, we will predict the **purchased variable (Dependent Variable)** by using **age and salary (Independent variables)**.



**Steps in Logistic Regression:** To implement the Logistic Regression using Python, we will use the same steps as we have done in previous topics of Regression. Below are the steps:

* Data Pre-processing step
* Fitting Logistic Regression to the Training set
* Predicting the test result
* Test accuracy of the result(Creation of Confusion matrix)
* Visualizing the test set result.

**Step 1: Data Pre-processing step**

In this step, we will pre-process/prepare the data so that we can use it in our code efficiently. It will be the same as we have done in Data pre-processing topic. The code for this is given below:

#Data Pre-procesing Step

# importing libraries

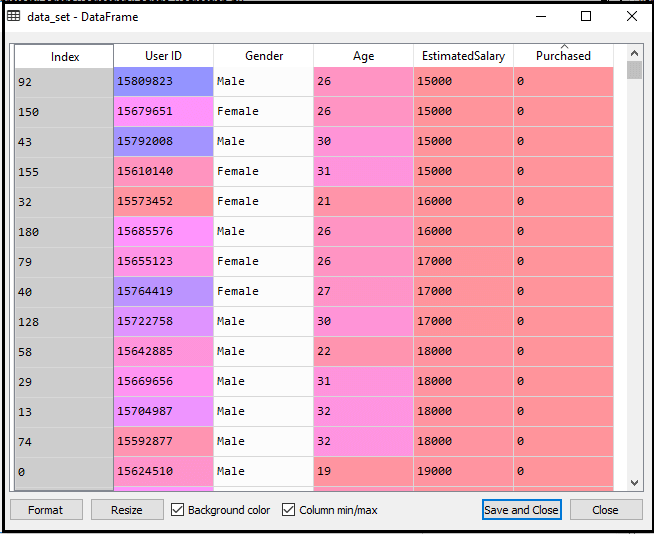
import numpy as nm

import matplotlib.pyplot as mtp

import pandas as pd

#importing datasets

data\_set= pd.read\_csv('user\_data.csv')

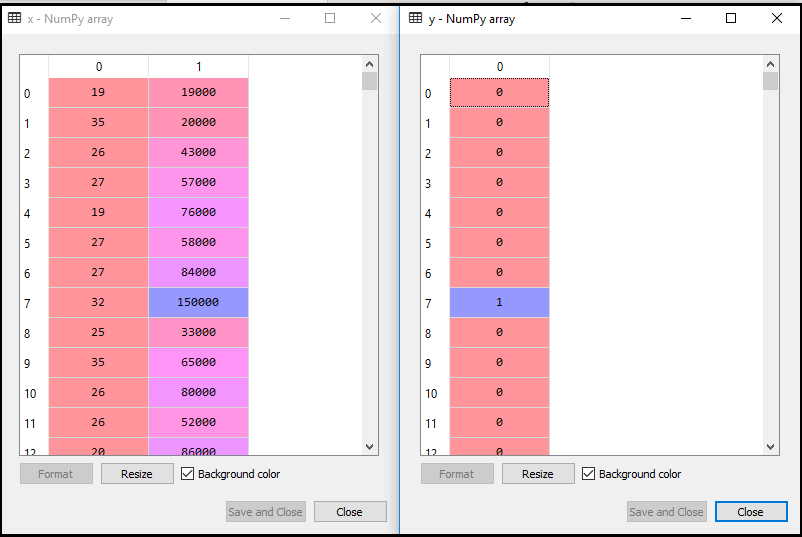


#Extracting Independent and dependent Variable

x= data\_set.iloc[:, [2,3]].values

y= data\_set.iloc[:, 4].values

output:

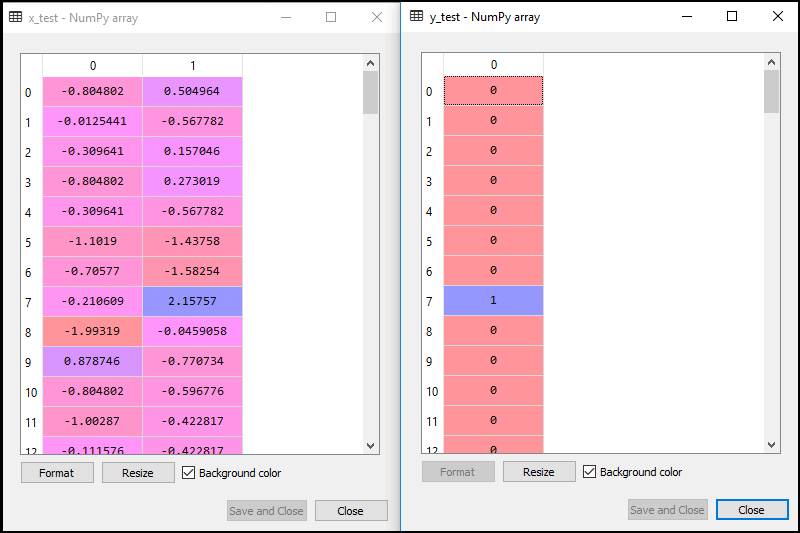


# Splitting the dataset into training and test set.

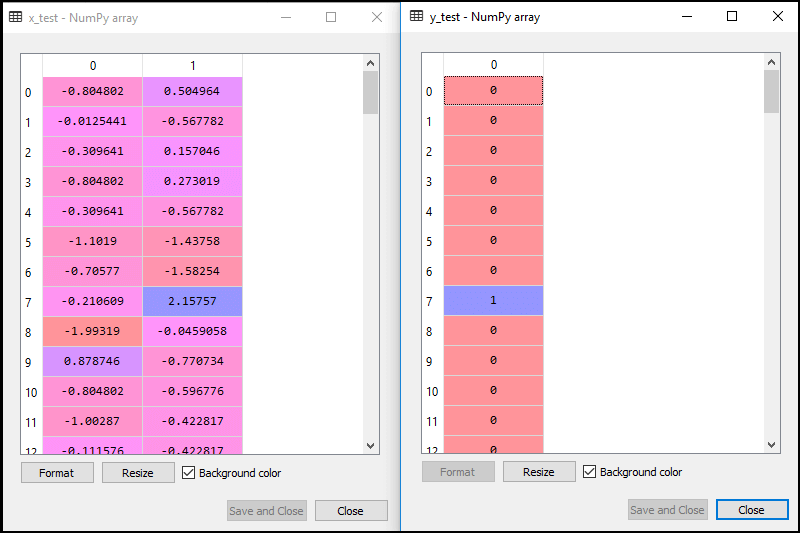
from sklearn.model\_selection import train\_test\_split

x\_train, x\_test, y\_train, y\_test= train\_test\_split(x, y, test\_size= 0.25, random\_state=0)

output: for test set



Output **For training set:**



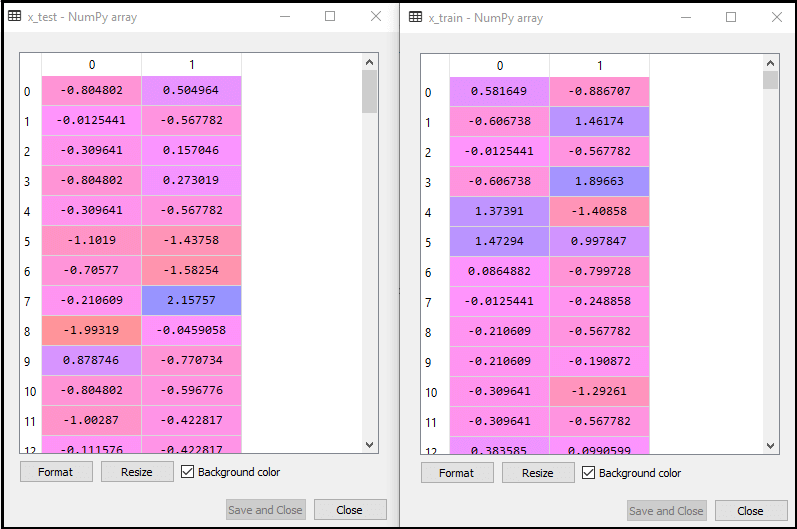
#feature Scaling

from sklearn.preprocessing import StandardScaler

st\_x= StandardScaler()

x\_train= st\_x.fit\_transform(x\_train)

x\_test= st\_x.transform(x\_test)



Step 2: **Fitting Logistic Regression to the Training set**

#Fitting Logistic Regression to the training set

from sklearn.linear\_model import LogisticRegression

classifier= LogisticRegression(random\_state=0)

classifier.fit(x\_train, y\_train)

**Out[5]:**

LogisticRegression(C=1.0, class\_weight=None, dual=False, fit\_intercept=True,

intercept\_scaling=1, l1\_ratio=None, max\_iter=100,

multi\_class='warn', n\_jobs=None, penalty='l2',

random\_state=0, solver='warn', tol=0.0001, verbose=0,

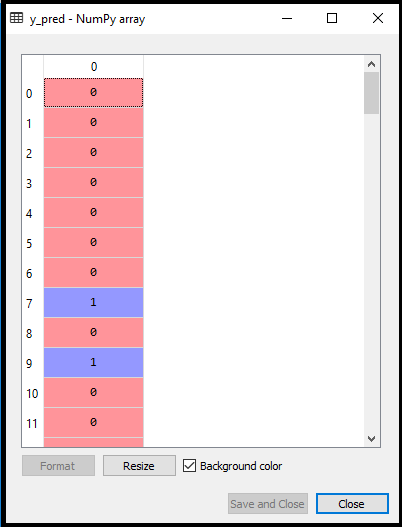
warm\_start=False)

step 3: **Predicting the Test Result**

#Predicting the test set result

y\_pred= classifier.predict(x\_test)

output:



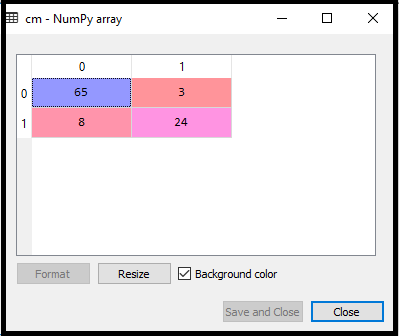
Step 4: **Test Accuracy of the result**

#Creating the Confusion matrix

from sklearn.metrics import confusion\_matrix

cm= confusion\_matrix()

output:



Step 5: **Visualizing the training set result**

#Visualizing the training set result

from matplotlib.colors import ListedColormap

x\_set, y\_set = x\_train, y\_train

x1, x2 = nm.meshgrid(nm.arange(start = x\_set[:, 0].min() - 1, stop = x\_set[:, 0].max() + 1, step =0.01),

nm.arange(start = x\_set[:, 1].min() - 1, stop = x\_set[:, 1].max() + 1, step = 0.01))

mtp.contourf(x1, x2, classifier.predict(nm.array([x1.ravel(), x2.ravel()]).T).reshape(x1.shape),

alpha = 0.75, cmap = ListedColormap(('purple','green' )))

mtp.xlim(x1.min(), x1.max())

mtp.ylim(x2.min(), x2.max())

for i, j in enumerate(nm.unique(y\_set)):

mtp.scatter(x\_set[y\_set == j, 0], x\_set[y\_set == j, 1],

c = ListedColormap(('purple', 'green'))(i), label = j)

mtp.title('Logistic Regression (Training set)')

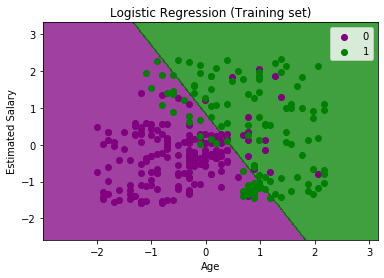
mtp.xlabel('Age')

mtp.ylabel('Estimated Salary')

mtp.legend()

mtp.show()

**Output:**



Step 6: **Visualizing the test set result**

#Visulaizing the test set result

from matplotlib.colors import ListedColormap

x\_set, y\_set = x\_test, y\_test

x1, x2 = nm.meshgrid(nm.arange(start = x\_set[:, 0].min() - 1, stop = x\_set[:, 0].max() + 1, step =0.01),

nm.arange(start = x\_set[:, 1].min() - 1, stop = x\_set[:, 1].max() + 1, step = 0.01))

mtp.contourf(x1, x2, classifier.predict(nm.array([x1.ravel(), x2.ravel()]).T).reshape(x1.shape),

alpha = 0.75, cmap = ListedColormap(('purple','green' )))

mtp.xlim(x1.min(), x1.max())

mtp.ylim(x2.min(), x2.max())

for i, j in enumerate(nm.unique(y\_set)):

mtp.scatter(x\_set[y\_set == j, 0], x\_set[y\_set == j, 1],

c = ListedColormap(('purple', 'green'))(i), label = j)

mtp.title('Logistic Regression (Test set)')

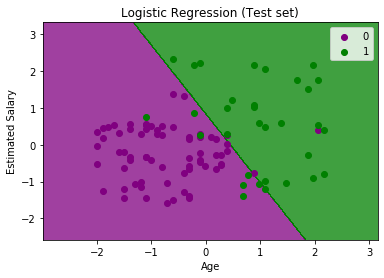
mtp.xlabel('Age')

mtp.ylabel('Estimated Salary')

mtp.legend()

mtp.show()

output:



**6. Random Forest (RF)**

**Implement and demonstrate a Random Forest classifier to classify the instances of dataset.**

**Display the classification results. Also, try the same algorithm to classify the instances for any given dataset.**

1.Data Pre-Processing Step:

# importing libraries

import numpy as nm

import matplotlib.pyplot as mtp

import pandas as pd

#importing datasets

data\_set= pd.read\_csv('user\_data.csv')

#Extracting Independent and dependent Variable

x= data\_set.iloc[:, [2,3]].values

y= data\_set.iloc[:, 4].values

# Splitting the dataset into training and test set.

from sklearn.model\_selection import train\_test\_split

x\_train, x\_test, y\_train, y\_test= train\_test\_split(x, y, test\_size= 0.25, random\_state=0)

#feature Scaling

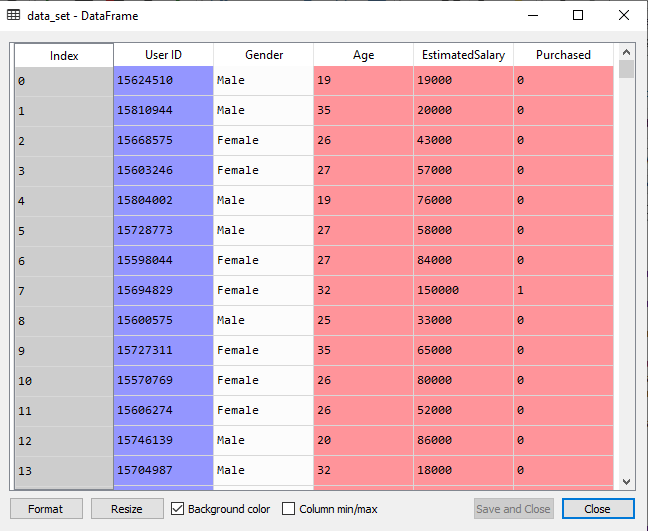
from sklearn.preprocessing import StandardScaler

st\_x= StandardScaler()

x\_train= st\_x.fit\_transform(x\_train)

x\_test= st\_x.transform(x\_test)

output:



2. Fitting the Random Forest algorithm to the training set:

#Fitting Decision Tree classifier to the training set

from sklearn.ensemble import RandomForestClassifier

classifier= RandomForestClassifier(n\_estimators= 10, criterion="entropy")

classifier.fit(x\_train, y\_train)

**output:**

**RandomForestClassifier(bootstrap=True, class\_weight=None, criterion='entropy',**

**max\_depth=None, max\_features='auto', max\_leaf\_nodes=None,**

**min\_impurity\_decrease=0.0, min\_impurity\_split=None,**

**min\_samples\_leaf=1, min\_samples\_split=2,**

**min\_weight\_fraction\_leaf=0.0, n\_estimators=10,**

**n\_jobs=None, oob\_score=False, random\_state=None,**

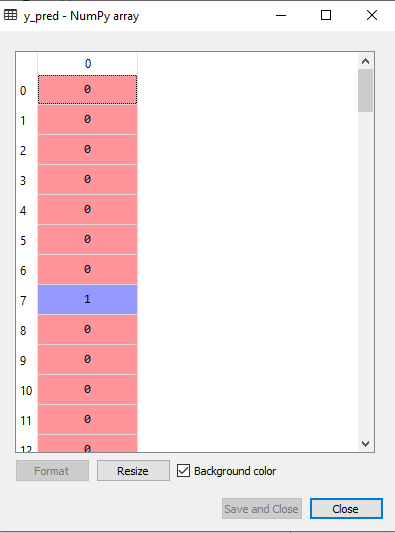
**verbose=0, warm\_start=False)**

3. Predicting the Test Set result

**#Predicting the test set result**

**y\_pred= classifier.predict(x\_test)**

**output:**



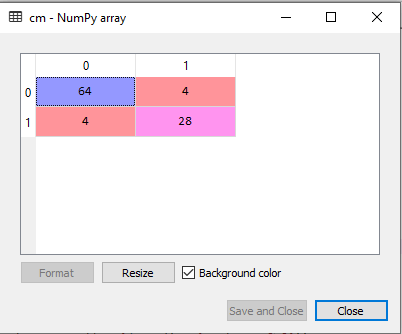
4. Creating the Confusion Matrix

**#Creating the Confusion matrix**

**from sklearn.metrics import confusion\_matrix**

**cm= confusion\_matrix(y\_test, y\_pred)**

**output:**



5. Visualizing the training Set result

from matplotlib.colors import ListedColormap

x\_set, y\_set = x\_train, y\_train

x1, x2 = nm.meshgrid(nm.arange(start = x\_set[:, 0].min() - 1, stop = x\_set[:, 0].max() + 1, step =0.01),

nm.arange(start = x\_set[:, 1].min() - 1, stop = x\_set[:, 1].max() + 1, step = 0.01))

mtp.contourf(x1, x2, classifier.predict(nm.array([x1.ravel(), x2.ravel()]).T).reshape(x1.shape),

alpha = 0.75, cmap = ListedColormap(('purple','green' )))

mtp.xlim(x1.min(), x1.max())

mtp.ylim(x2.min(), x2.max())

for i, j in enumerate(nm.unique(y\_set)):

mtp.scatter(x\_set[y\_set == j, 0], x\_set[y\_set == j, 1],

c = ListedColormap(('purple', 'green'))(i), label = j)

mtp.title('Random Forest Algorithm (Training set)')

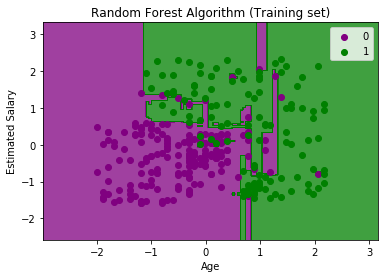
mtp.xlabel('Age')

mtp.ylabel('Estimated Salary')

mtp.legend()

mtp.show()

output:



6. Visualizing the test set result

#Visulaizing the test set result

from matplotlib.colors import ListedColormap

x\_set, y\_set = x\_test, y\_test

x1, x2 = nm.meshgrid(nm.arange(start = x\_set[:, 0].min() - 1, stop = x\_set[:, 0].max() + 1, step =0.01),

nm.arange(start = x\_set[:, 1].min() - 1, stop = x\_set[:, 1].max() + 1, step = 0.01))

mtp.contourf(x1, x2, classifier.predict(nm.array([x1.ravel(), x2.ravel()]).T).reshape(x1.shape),

alpha = 0.75, cmap = ListedColormap(('purple','green' )))

mtp.xlim(x1.min(), x1.max())

mtp.ylim(x2.min(), x2.max())

for i, j in enumerate(nm.unique(y\_set)):

mtp.scatter(x\_set[y\_set == j, 0], x\_set[y\_set == j, 1],

c = ListedColormap(('purple', 'green'))(i), label = j)

mtp.title('Random Forest Algorithm(Test set)')

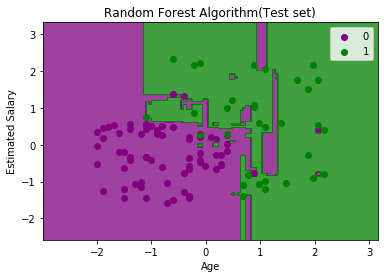
mtp.xlabel('Age')

mtp.ylabel('Estimated Salary')

mtp.legend()

mtp.show()

output:



**6. K-Means Clustering**

**Implement and demonstrate the k-means clustering algorithms. Visualize the results using graphs.**

The steps to be followed for the implementation are given below:

* **Data Pre-processing**
* **Finding the optimal number of clusters using the elbow method**
* **Training the K-means algorithm on the training dataset**
* **Visualizing the clusters**

Step-1: Data pre-processing Step

# importing libraries

import numpy as nm

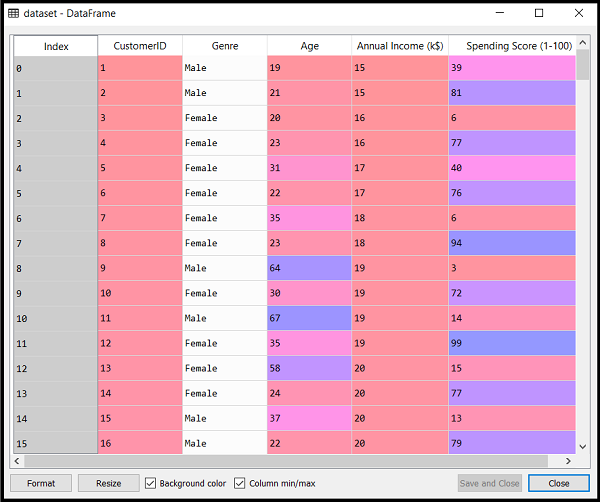
import matplotlib.pyplot as mtp

import pandas as pd

# Importing the dataset

dataset = pd.read\_csv('Mall\_Customers\_data.csv')

output:



**Extracting Independent Variables**

x = dataset.iloc[:, [3, 4]].values

As we can see, we are extracting only 3rd and 4th feature. It is because we need a 2d plot to visualize the model, and some features are not required, such as customer\_id.

Step-2: Finding the optimal number of clusters using the elbow method

#finding optimal number of clusters using the elbow method

from sklearn.cluster import KMeans

wcss\_list= [] #Initializing the list for the values of WCSS

#Using for loop for iterations from 1 to 10.

for i in range(1, 11):

kmeans = KMeans(n\_clusters=i, init='k-means++', random\_state= 42)

kmeans.fit(x)

wcss\_list.append(kmeans.inertia\_)

mtp.plot(range(1, 11), wcss\_list)

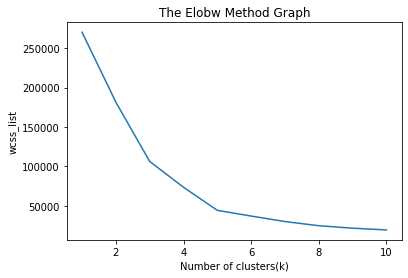
mtp.title('The Elobw Method Graph')

mtp.xlabel('Number of clusters(k)')

mtp.ylabel('wcss\_list')

mtp.show()

**output:**



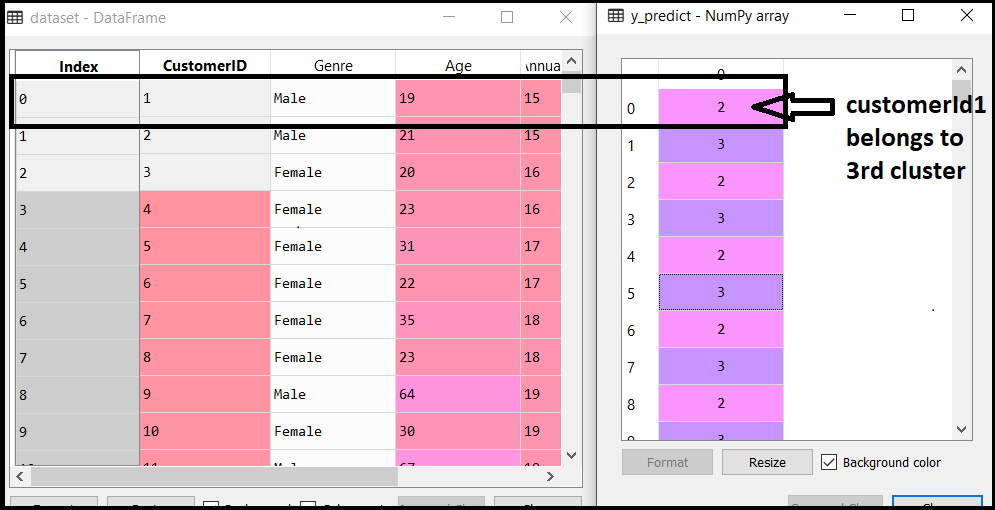
Step- 3: Training the K-means algorithm on the training dataset

#training the K-means model on a dataset

kmeans = KMeans(n\_clusters=5, init='k-means++', random\_state= 42)

y\_predict= kmeans.fit\_predict(x)

y\_predict



Step-4: Visualizing the Clusters

#visulaizing the clusters

mtp.scatter(x[y\_predict == 0, 0], x[y\_predict == 0, 1], s = 100, c = 'blue', label = 'Cluster 1') #for first cluster

mtp.scatter(x[y\_predict == 1, 0], x[y\_predict == 1, 1], s = 100, c = 'green', label = 'Cluster 2') #for second cluster

mtp.scatter(x[y\_predict== 2, 0], x[y\_predict == 2, 1], s = 100, c = 'red', label = 'Cluster 3') #for third cluster

mtp.scatter(x[y\_predict == 3, 0], x[y\_predict == 3, 1], s = 100, c = 'cyan', label = 'Cluster 4') #for fourth cluster

mtp.scatter(x[y\_predict == 4, 0], x[y\_predict == 4, 1], s = 100, c = 'magenta', label = 'Cluster 5') #for fifth cluster

mtp.scatter(kmeans.cluster\_centers\_[:, 0], kmeans.cluster\_centers\_[:, 1], s = 300, c = 'yellow', label = 'Centroid')

mtp.title('Clusters of customers')

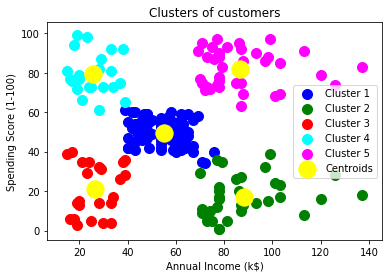
mtp.xlabel('Annual Income (k$)')

mtp.ylabel('Spending Score (1-100)')

mtp.legend()

mtp.show()

output:



**8. Hierarchical clustering**

**Implement and demonstrate the hierarchical clustering algorithms. Visualize the results using graphs.**

Steps for implementation of AHC using Python:

The steps for implementation will be the same as the k-means clustering, except for some changes such as the method to find the number of clusters. Below are the steps:

1. **Data Pre-processing**
2. **Finding the optimal number of clusters using the Dendrogram**
3. **Training the hierarchical clustering model**
4. **Visualizing the clusters**

### Step 1: Data Pre-processing Steps:

**Importing the libraries**

# Importing the libraries

import numpy as nm

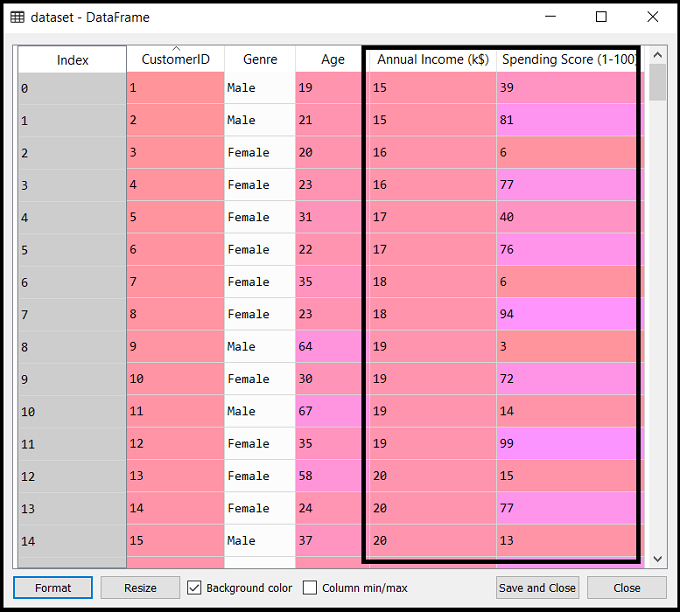
import matplotlib.pyplot as mtp

import pandas as pd

**Importing the dataset**

# Importing the dataset

dataset = pd.read\_csv('Mall\_Customers\_data.csv')



**Extracting the matrix of features**

x = dataset.iloc[:, [3, 4]].values

Step-2: Finding the optimal number of clusters using the Dendrogram

#Finding the optimal number of clusters using the dendrogram

import scipy.cluster.hierarchy as shc

dendro = shc.dendrogram(shc.linkage(x, method="ward"))

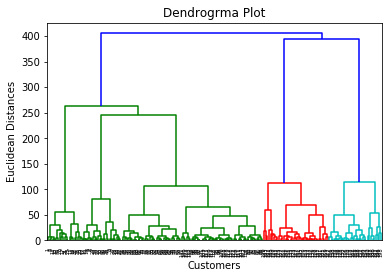
mtp.title("Dendrogrma Plot")

mtp.ylabel("Euclidean Distances")

mtp.xlabel("Customers")

mtp.show()

output:



Step-3: Training the hierarchical clustering model

#training the hierarchical model on dataset

from sklearn.cluster import AgglomerativeClustering

hc= AgglomerativeClustering(n\_clusters=5, affinity='euclidean', linkage='ward')

y\_pred= hc.fit\_predict(x)

output:



Step-4: Visualizing the clusters

#visulaizing the clusters

mtp.scatter(x[y\_pred == 0, 0], x[y\_pred == 0, 1], s = 100, c = 'blue', label = 'Cluster 1')

mtp.scatter(x[y\_pred == 1, 0], x[y\_pred == 1, 1], s = 100, c = 'green', label = 'Cluster 2')

mtp.scatter(x[y\_pred== 2, 0], x[y\_pred == 2, 1], s = 100, c = 'red', label = 'Cluster 3')

mtp.scatter(x[y\_pred == 3, 0], x[y\_pred == 3, 1], s = 100, c = 'cyan', label = 'Cluster 4')

mtp.scatter(x[y\_pred == 4, 0], x[y\_pred == 4, 1], s = 100, c = 'magenta', label = 'Cluster 5')

mtp.title('Clusters of customers')

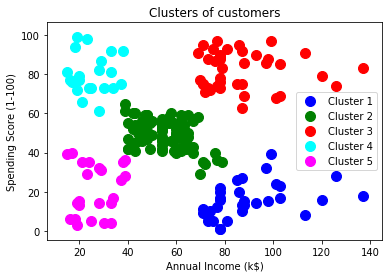
mtp.xlabel('Annual Income (k$)')

mtp.ylabel('Spending Score (1-100)')

mtp.legend()

mtp.show()

output:



**9. DBSCAN clustering**

**Implement and demonstrate the hierarchical clustering algorithms. Visualize the results using graphs.**

We have to follow the following steps in order to implement the DBSCAN algorithm and its logic inside a Python program:

**Step 1: Importing all the required libraries:**

# Importing numpy library as nmp

import numpy as nmp

# Importing pandas library as pds

import pandas as pds

# Importing matplotlib library as pplt

import matplotlib.pyplot as pplt

# Importing DBSCAN from cluster module of Sklearn library

from sklearn.cluster import DBSCAN

# Importing StandardSclaer and normalize from preprocessing module of Sklearn library

from sklearn.preprocessing import StandardScaler

from sklearn.preprocessing import normalize

# Importing PCA from decomposition module of Sklearn

from sklearn.decomposition import PCA

**Step 2: Loading the Data:**

# Loading the data inside an initialized variable

M = pds.read\_csv('sampleDataset.csv') # Path of dataset file

# Dropping the CUST\_ID column from the dataset with drop() function

M = M.drop('CUST\_ID', axis = 1)

# Using fillna() function to handle missing values

M.fillna(method ='ffill', inplace = True)

# Printing dataset head in output

print(M.head())

output:

BALANCE BALANCE\_FREQUENCY ... PRC\_FULL\_PAYMENT TENURE

0 40.900749 0.818182 ... 0.000000 12

1 3202.467416 0.909091 ... 0.222222 12

2 2495.148862 1.000000 ... 0.000000 12

3 1666.670542 0.636364 ... 0.000000 12

4 817.714335 1.000000 ... 0.000000 12

[5 rows x 17 columns]

**Step 3: Preprocessing the data:**

# Initializing a variable with the StandardSclaer() function

scalerFD = StandardScaler()

# Transforming the data of dataset with Scaler

M\_scaled = scalerFD.fit\_transform(M)

# To make sure that data will follow gaussian distribution

# We will normalize the scaled data with normalize() function

M\_normalized = normalize(M\_scaled)

# Now we will convert numpy arrays in the dataset into dataframes of panda

M\_normalized = pds.DataFrame(M\_normalized)

**Step 4: Reduce the dimensionality of the data:**

# Initializing a variable with the PCA() function

pcaFD = PCA(n\_components = 2) # components of data

# Transforming the normalized data with PCA

M\_principal = pcaFD.fit\_transform(M\_normalized)

# Making dataframes from the transformed data

M\_principal = pds.DataFrame(M\_principal)

# Creating two columns in the transformed data

M\_principal.columns = ['C1', 'C2']

# Printing the head of the transformed data

print(M\_principal.head())

output:

C1 C2

0 -0.489949 -0.679976

1 -0.519099 0.544828

2 0.330633 0.268877

3 -0.481656 -0.097610

4 -0.563512 -0.482506

**Step 5: Build a clustering model:**

# Creating clustering model of the data using the DBSCAN function and providing parameters in it

db\_default = DBSCAN(eps = 0.0375, min\_samples = 3).fit(M\_principal)

# Labelling the clusters we have created in the dataset

labeling = db\_default.labels\_

**Step 6: Visualize the clustering model:**

# Visualization of clustering model by giving different colours

colours = {}

# First colour in visualization is green

colours[0] = 'g'

# Second colour in visualization is black

colours[1] = 'k'

# Third colour in visualization is red

colours[2] = 'r'

# Last colour in visualization is blue

colours[-1] = 'b'

# Creating a colour vector for each data point in the dataset cluster

cvec = [colours[label] for label in labeling]

# Construction of the legend

# Scattering of green colour

g = pplt.scatter(M\_principal['C1'], M\_principal['C2'], color ='g');

# Scattering of black colour

k = pplt.scatter(M\_principal['C1'], M\_principal['C2'], color ='k');

# Scattering of red colour

r = pplt.scatter(M\_principal['C1'], M\_principal['C2'], color ='r');

# Scattering of green colour

b = pplt.scatter(M\_principal['C1'], M\_principal['C2'], color ='b');

# Plotting C1 column on the X-Axis and C2 on the Y-Axis

# Fitting the size of the figure with figure function

pplt.figure(figsize =(9, 9))

# Scattering the data points in the Visualization graph

pplt.scatter(M\_principal['C1'], M\_principal['C2'], c = cvec)

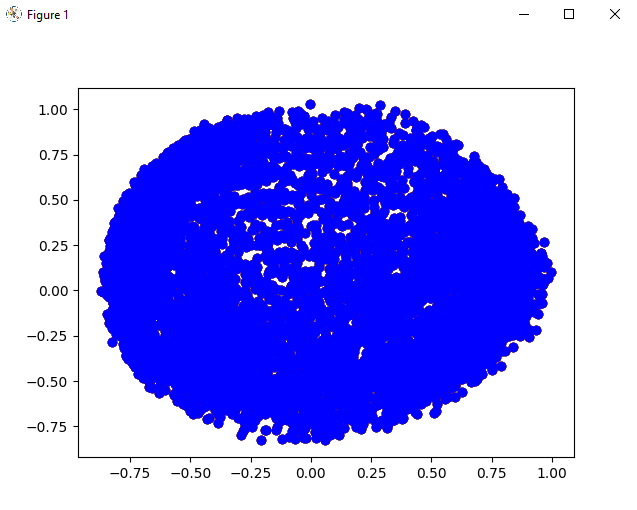
# Building the legend with the coloured data points and labelled

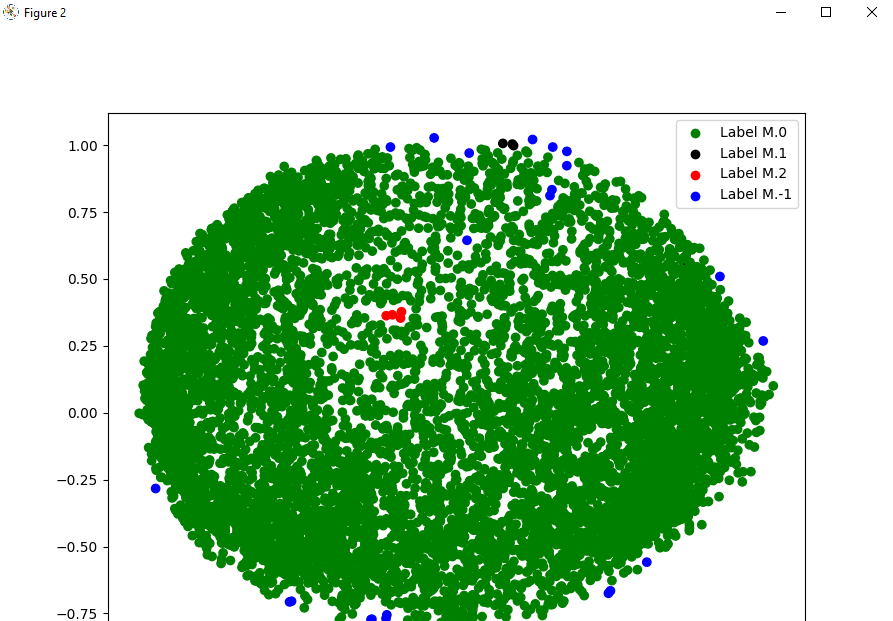
pplt.legend((g, k, r, b), ('Label M.0', 'Label M.1', 'Label M.2', 'Label M.-1'))

# Showing Visualization in the output

pplt.show()

output:





**Step 7: Tuning the parameters:**

# Tuning the parameters of the model inside the DBSCAN function

dts = DBSCAN(eps = 0.0375, min\_samples = 50).fit(M\_principal)

# Labelling the clusters of data points

labeling = dts.labels\_

**Step 8: Visualization of the changes:**

# Labelling with different colours

colours1 = {}

# labelling with Red colour

colours1[0] = 'r'

# labelling with Green colour

colours1[1] = 'g'

# labelling with Blue colour

colours1[2] = 'b'

colours1[3] = 'c'

# labelling with Yellow colour

colours1[4] = 'y'

# Magenta colour

colours1[5] = 'm'

# labelling with Black colour

colours1[-1] = 'k'

# Labelling the data points with the colour variable we have defined

cvec = [colours1[label] for label in labeling]

# Defining all colour that we will use

colors = ['r', 'g', 'b', 'c', 'y', 'm', 'k' ]

# Scattering the colours onto the data points

r = pplt.scatter(

M\_principal['C1'], M\_principal['C2'], marker ='o', color = colors[0])

g = pplt.scatter(

M\_principal['C1'], M\_principal['C2'], marker ='o', color = colors[1])

b = pplt.scatter(

M\_principal['C1'], M\_principal['C2'], marker ='o', color = colors[2])

c = pplt.scatter(

M\_principal['C1'], M\_principal['C2'], marker ='o', color = colors[3])

y = pplt.scatter(

M\_principal['C1'], M\_principal['C2'], marker ='o', color = colors[4])

m = pplt.scatter(

M\_principal['C1'], M\_principal['C2'], marker ='o', color = colors[5])

k = pplt.scatter(

M\_principal['C1'], M\_principal['C2'], marker ='o', color = colors[6])

# Fitting the size of the figure with figure function

pplt.figure(figsize =(9, 9))

# Scattering column 1 into X-axis and column 2 into y-axis

pplt.scatter(M\_principal['C1'], M\_principal['C2'], c = cvec)

# Constructing a legend with the colours we have defined

pplt.legend((r, g, b, c, y, m, k),

('Label M.0', 'Label M.1', 'Label M.2', 'Label M.3', 'Label M.4','Label M.5', 'Label M.-1'), # Using different labels for data points

scatterpoints = 1, # Defining the scatter point

loc ='upper left', # Location of cluster scattering

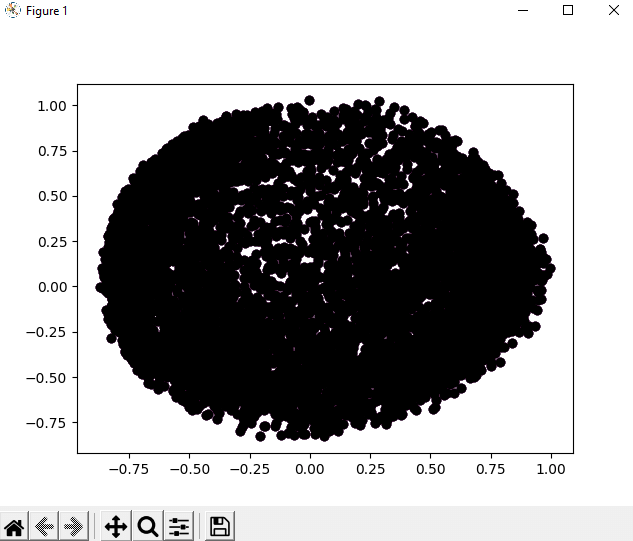
ncol = 3, # Number of columns

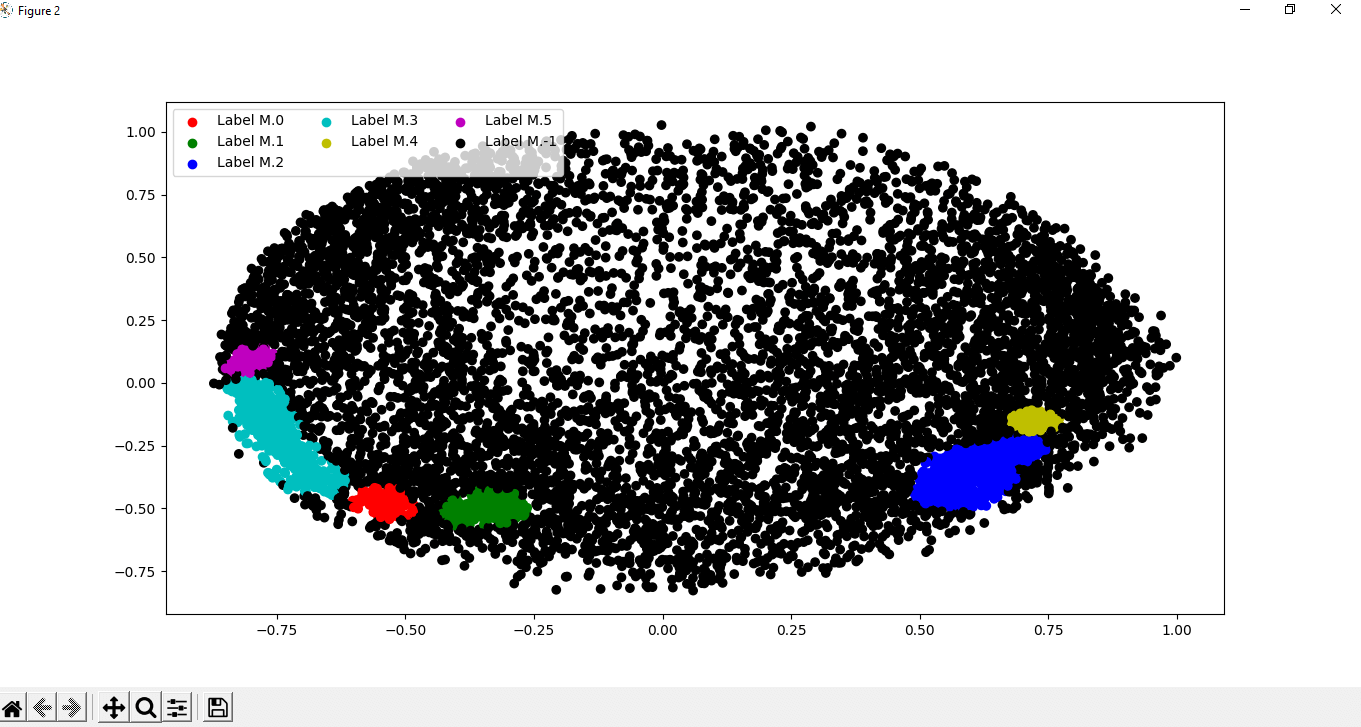
fontsize = 10) # Size of the font

# Displaying the visualisation of changes in cluster scattering

pplt.show()

output:





**10. Implement and demonstrate the two hidden layer multilayer perceptron neural network to any given dataset for classification. Apply two different optimizers or activation functions and compare the results.**

#Importing the essential modules in the hidden layer

import tensorflow as tf

import numpy as np

import mat

plotlib.pyplot as plt

import math, random

np.random.seed(1000)

function\_to\_learn = lambda x: np.cos(x) + 0.1\*np.random.randn(\*x.shape)

layer\_1\_neurons = 10

NUM\_points = 1000

#Train the parameters of hidden layer

batch\_size = 100

NUM\_EPOCHS = 1500

all\_x = np.float32(np.random.uniform(-2\*math.pi, 2\*math.pi, (1, NUM\_points))).T

np.random.shuffle(all\_x)

train\_size = int(900)

#Train the first 700 points in the set x\_training = all\_x[:train\_size]

y\_training = function\_to\_learn(x\_training)

#Training the last 300 points in the given set x\_validation = all\_x[train\_size:]

y\_validation = function\_to\_learn(x\_validation)

plt.figure(1)

plt.scatter(x\_training, y\_training, c = 'blue', label = 'train')

plt.scatter(x\_validation, y\_validation, c = 'pink', label = 'validation')

plt.legend()

plt.show()

X = tf.placeholder(tf.float32, [None, 1], name = "X")

Y = tf.placeholder(tf.float32, [None, 1], name = "Y")

#first layer

#Number of neurons = 10

w\_h = tf.Variable(

tf.random\_uniform([1, layer\_1\_neurons],\ minval = -1, maxval = 1, dtype = tf.float32))

b\_h = tf.Variable(tf.zeros([1, layer\_1\_neurons], dtype = tf.float32))

h = tf.nn.sigmoid(tf.matmul(X, w\_h) + b\_h)

#output layer

#Number of neurons = 10

w\_o = tf.Variable(

tf.random\_uniform([layer\_1\_neurons, 1],\ minval = -1, maxval = 1, dtype = tf.float32))

b\_o = tf.Variable(tf.zeros([1, 1], dtype = tf.float32))

#building the model

model = tf.matmul(h, w\_o) + b\_o

#minimize the cost function (model - Y)

train\_op = tf.train.AdamOptimizer().minimize(tf.nn.l2\_loss(model - Y))

#Starting the Learning phase

sess = tf.Session() sess.run(tf.initialize\_all\_variables())

errors = []

for i in range(NUM\_EPOCHS):

for start, end in zip(range(0, len(x\_training), batch\_size),\

range(batch\_size, len(x\_training), batch\_size)):

sess.run(train\_op, feed\_dict = {X: x\_training[start:end],\ Y: y\_training[start:end]})

cost = sess.run(tf.nn.l2\_loss(model - y\_validation),\ feed\_dict = {X:x\_validation})

errors.append(cost)

if i%100 == 0:

print("epoch %d, cost = %g" % (i, cost))

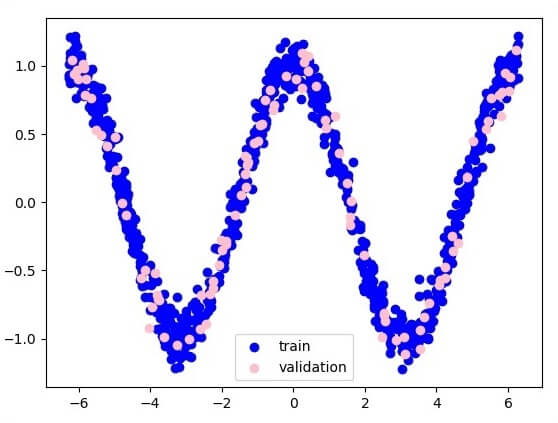
plt.plot(errors,label='MLP Function Approximation') plt.xlabel('epochs')

plt.ylabel('cost')

plt.legend()

plt.show()

output:



The two data are: **train** and **validation**, which are described in distinct colors as visible in the legend section.

